Group A

1. Create an array to store student names in a class, allowing insertion, deletion, and traversal operations.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string students[50];  // Array to store student names
    int n = 0;            // Current number of students
    int choice;
  do {
        cout << "\n----- Student Array Operations -----";
        cout << "\n1. Insert Student";
        cout << "\n2. Delete Student";
        cout << "\n3. Display Students";
        cout << "\n4. Exit";
        cout << "\nEnter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1: {
                if (n >= 50) {
                    cout << "Array is full! Cannot insert more students.\n";
                    break;
```

```
    string name;
    cout << "Enter student name to insert: ";
    cin >> ws; // clear input buffer
    getline(cin, name);
    students[n++] = name;
    cout << name << " added successfully!\n";
    break;
}
  case 2: {
    if (n == 0) {
        cout << "No students to delete.\n";
        break;
    }
    string name;
    cout << "Enter student name to delete: ";
    cin >> ws;
    getline(cin, name);
   bool found = false;
    for (int i = 0; i < n; i++) {
        if (students[i] == name) {
            // Shift elements left
            for (int j = i; j < n - 1; j++) {
                students[j] = students[j + 1];
            }
            n--;
```

```
            found = true;
            cout << name << " deleted successfully!\n";
            break;
        }
    }
    if (!found)
        cout << "Student not found.\n";
    break;
}
   case 3: {
    if (n == 0) {
        cout << "No students in the list.\n";
        break;
    }
    cout << "\nList of Students:\n";
    for (int i = 0; i < n; i++) {
        cout << i + 1 << ". " << students[i] << endl;
    }
    break;
}
  case 4:
    cout << "Exiting program.\n";
    break;
   default:
    cout << "Invalid choice. Try again.\n";  }
```

```
    } while (choice != 4);
        return 0;
}
```

Output :

----- Student Array Operations -----

1. Insert Student

2. Delete Student

3. Display Students

4. Exit

Enter your choice: 1

Enter student name to insert: Rahul

Rahul added successfully!

----- Student Array Operations -----

Enter your choice: 1

Enter student name to insert: Priya

Priya added successfully!

----- Student Array Operations -----

Enter your choice: 3

List of Students:

1. Rahul

2. Priya

----- Student Array Operations -----

Enter your choice: 2

Enter student name to delete: Rahul

Rahul deleted successfully!

----- Student Array Operations -----

Enter your choice: 3

List of Students:

1. Priya

----- Student Array Operations -----

Enter your choice: 4

Exiting program.

2. Store and display marks of students in multiple subjects using a 2D array.

```
#include <iostream>
using namespace std;
int main() {
    int students, subjects;
    cout << "Enter number of students: ";
    cin >> students;
    cout << "Enter number of subjects: ";
    cin >> subjects;
 int marks[50][50];  // 2D array to store marks
  // Input marks
    cout << "\nEnter marks of students:\n";
    for (int i = 0; i < students; i++) {
        cout << "\nStudent " << i + 1 << ":\n";
        for (int j = 0; j < subjects; j++) {
            cout << "Enter marks for subject " << j + 1 << ": ";
            cin >> marks[i][j];
        }
    }
  // Display marks
    cout << "\n----- Marks of Students -----\n";
    for (int i = 0; i < students; i++) {
        cout << "Student " << i + 1 << ": ";
        for (int j = 0; j < subjects; j++) {
            cout << marks[i][j] << "\t"
```

```
 }
        cout << endl;
    }
  return 0;
}
```

Output :

Enter number of students: 3

Enter number of subjects: 3

Enter marks of students:

Student 1:

Enter marks for subject 1: 85

Enter marks for subject 2: 78

Enter marks for subject 3: 90

Student 2:

Enter marks for subject 1: 76

Enter marks for subject 2: 88

Enter marks for subject 3: 69

Student 3:

Enter marks for subject 1: 92

Enter marks for subject 2: 81

Enter marks for subject 3: 79

----- Marks of Students -----

Student 1: 85 78 90

Student 2: 76 88 69

Student 3: 92 81 79

Group B

1. Implement a student record system using a singly linked list that supports creating, inserting, deleting, and displaying student records.

```
#include <iostream>
#include <string>
using namespace std;
// Structure to represent a student node
struct Student {
    int rollNo;
    string name;
    float marks;
    Student* next;
};
// Function to create a new student node
Student* createStudent(int roll, string name, float marks) {
    Student* newStudent = new Student;
    newStudent->rollNo = roll;
    newStudent->name = name;
    newStudent->marks = marks;
    newStudent->next = nullptr;
    return newStudent;
}
// Insert student at the end
void insertStudent(Student*& head, int roll, string name, float marks) {
    Student* newStudent = createStudent(roll, name, marks);
```

```
  if (head == nullptr) {
        head = newStudent;
    } else {
        Student* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newStudent;
    }
    cout << " Student record inserted successfully.\n";
}
// Delete student by roll number
void deleteStudent(Student*& head, int roll) {
    if (head == nullptr) {
        cout << " No records to delete.\n";
        return;
    }
 // If the first node is to be deleted
    if (head->rollNo == roll) {
        Student* temp = head;
        head = head->next;
        delete temp;
        cout << " Record deleted successfully.\n";
        return;
    }
```

```
// Search for the node to delete
    Student* temp = head;
    Student* prev = nullptr;
    while (temp != nullptr && temp->rollNo != roll) {
        prev = temp;
        temp = temp->next;
    }
  if (temp == nullptr) {
        cout << "Record not found.\n";
    } else {
        prev->next = temp->next;
        delete temp;
        cout << " Record deleted successfully.\n";
    }
}
// Display all student records
void displayStudents(Student* head) {
    if (head == nullptr) {
        cout << " No student records found.\n";
        return;
    }
    cout << "\n----- Student Records -----\n";
    while (head != nullptr) {
        cout << "Roll No: " << head->rollNo
             << ", Name: " << head->name
```

```
            << ", Marks: " << head->marks << endl;
        head = head->next;
    }
}
int main() {
    Student* head = nullptr;
    int choice, roll;
    string name;
    float marks;
    do {
        cout << "\n----- Student Record System (Linked List) -----";
        cout << "\n1. Insert Student Record";
        cout << "\n2. Delete Student Record";
        cout << "\n3. Display All Records";
        cout << "\n4. Exit";
        cout << "\nEnter your choice: ";
        cin >> choice;
        cin.ignore();
       switch (choice) {
            case 1:
                cout << "Enter Roll No: ";
                cin >> roll;
                cin.ignore();
                cout << "Enter Name: ";
                getline(cin, name);
```

```
            cout << "Enter Marks: ";
            cin >> marks;
            insertStudent(head, roll, name, marks);
            break;
        case 2:
            cout << "Enter Roll No to delete: ";
            cin >> roll;
            deleteStudent(head, roll);
            break;
        case 3:
            displayStudents(head);
            break;
        case 4:
            cout << "Exiting program...\n";
            break;
        default:
            cout << "Invalid choice. Try again.\n";
      }
 } while (choice != 4);
return 0;
}
```

Output :

----- Student Record System (Linked List) -----

1. Insert Student Record

2. Delete Student Record

3. Display All Records

4. Exit

Enter your choice: 1

Enter Roll No: 101

Enter Name: Riya

Enter Marks: 85

Student record inserted successfully.

Enter your choice: 1

Enter Roll No: 102

Enter Name: Rahul

Enter Marks: 78

Student record inserted successfully.

Enter your choice: 3

----- Student Records -----

Roll No: 101, Name: Riya, Marks: 85

Roll No: 102, Name: Rahul, Marks: 78

Enter your choice: 2

Enter Roll No to delete: 101

Record deleted successfully.

Enter your choice: 3

----- Student Records -----

Roll No: 102, Name: Rahul, Marks: 78

Enter your choice: 4

Exiting program...

2. Merge two linked lists containing student names from two different classes into a single list.

```
#include <iostream>
#include <string>
using namespace std;
// Node structure for linked list
struct Node {
    string name;
    Node* next;
};
// Function to create a new node
Node* createNode(string name) {
    Node* newNode = new Node;
    newNode->name = name;
    newNode->next = nullptr;
    return newNode;
}
// Function to insert a student name at the end of the list
void insertStudent(Node*& head, string name) {
    Node* newNode = createNode(name);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
```

```
        }
        temp->next = newNode;
    }
}
// Function to display the list of students
void displayList(Node* head) {
    if (head == nullptr) {
        cout << "No students in the list.\n";
        return;
    }
     while (head != nullptr) {
        cout << head->name << " -> ";
        head = head->next;
    }
    cout << "NULL\n";
}
// Function to merge two linked lists
Node* mergeLists(Node* list1, Node* list2) {
    if (list1 == nullptr) return list2;
    if (list2 == nullptr) return list1;
    Node* temp = list1;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = list2;
```

```
    return list1;
}
int main() {
    Node* classA = nullptr;
    Node* classB = nullptr;
    Node* mergedList = nullptr;
    int n1, n2;
    string name;
    cout << "Enter number of students in Class A: ";
    cin >> n1;
    cin.ignore();
    cout << "Enter names of Class A students:\n";
    for (int i = 0; i < n1; i++) {
        cout << "Student " << i + 1 << ": ";
        getline(cin, name);
        insertStudent(classA, name);
    }
   cout << "\nEnter number of students in Class B: ";
    cin >> n2;
    cin.ignore();
     cout << "Enter names of Class B students:\n";
    for (int i = 0; i < n2; i++) {
        cout << "Student " << i + 1 << ": ";
        getline(cin, name);
```

```
        insertStudent(classB, name);
    }
    cout << "\nClass A List: ";
    displayList(classA);
    cout << "Class B List: ";
    displayList(classB);
    // Merge both lists
    mergedList = mergeLists(classA, classB);
    cout << "\nMerged Student List: ";
    displayList(mergedList);
    return 0;
}
```

Output :

Enter number of students in Class A: 3

Enter names of Class A students:

Student 1: Riya

Student 2: Aarav

Student 3: Meena

Enter number of students in Class B: 2

Enter names of Class B students:

Student 1: Rahul

Student 2: Priya

Class A List: Riya -> Aarav -> Meena -> NULL

Class B List: Rahul -> Priya -> NULL

Merged Student List: Riya -> Aarav -> Meena -> Rahul -> Priya -> NULL

Group C

1. Use the stack ADT to check whether parentheses in an arithmetic expression are balanced

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
// Function to check if parentheses are balanced
bool isBalanced(string expr) {
    stack<char> s;
    for (char ch : expr) {
        // Push opening brackets
        if (ch == '(' || ch == '{' || ch == '[') {
            s.push(ch);
        }
        // Check closing brackets
        else if (ch == ')' || ch == '}' || ch == ']') {
            if (s.empty()) return false; // No matching opening bracket
            char top = s.top();
            s.pop();
           // Check for matching pair
            if ((ch == ')' && top != '(') ||
                (ch == '}' && top != '{') ||
```

```
            (ch == ']' && top != '[')) {
            return false;
        }
      }
   }
// If stack is empty, all brackets are balanced
   return s.empty();
}
int main() {
   string expression;
   cout << "Enter an arithmetic expression: ";
   getline(cin, expression);
  if (isBalanced(expression))
      cout << " Parentheses are balanced.\n";
   else
      cout << " Parentheses are NOT balanced.\n";
    return 0;
}
```

Output: 1

Enter an arithmetic expression: (a+b)*(c+d)

Parentheses are balanced.

Output : 2

Enter an arithmetic expression: (a+b*(c-d)

Parentheses are NOT balanced.

2. Implement a queue to simulate a customer service system where customers join and leave in FIFO order.

```
#include <iostream>
#include <queue>
#include <string>
using namespace std;
int main() {
    queue<string> customerQueue;  // Queue to store customer names
    int choice;
    string name;
 do {
        cout << "\n----- Customer Service System -----";
        cout << "\n1. Add Customer to Queue";
        cout << "\n2. Serve (Remove) Customer";
        cout << "\n3. Display Waiting Customers";
        cout << "\n4. Exit";
        cout << "\nEnter your choice: ";
        cin >> choice;
        cin.ignore(); // Clear input buffer
       switch (choice) {
            case 1:
                cout << "Enter customer name: ";
                getline(cin, name);
                customerQueue.push(name);
                cout << name << " has joined the queue.\n";
```

```
            break;


        case 2:
            if (customerQueue.empty()) {
                cout << "No customers to serve.\n";
            } else {
                cout << customerQueue.front() << " has been served and removed from the queue.\n";
                customerQueue.pop();
            }
            break;
        case 3:
            if (customerQueue.empty()) {
                cout << "No customers in the queue.\n";
            } else {
                cout << "\nCustomers currently waiting:\n";
                queue<string> temp = customerQueue; // Temporary copy to display all customers
                int position = 1;
                while (!temp.empty()) {
                    cout << position++ << ". " << temp.front() << endl;
                    temp.pop();
                }
            }
            break;
        case 4:
```

```
            cout << "Exiting Customer Service System.\n";
            break;
          default:
            cout << "Invalid choice. Please try again.\n";
      }
  } while (choice != 4);
              return 0;
}
```

Output :

----- Customer Service System -----

1. Add Customer to Queue

2. Serve (Remove) Customer

3. Display Waiting Customers

4. Exit

Enter your choice: 1

Enter customer name: Rohan

Rohan has joined the queue.

Enter your choice: 1

Enter customer name: Priya

Priya has joined the queue.

Enter your choice: 3

Customers currently waiting:

1. Rohan

2. Priya

Enter your choice: 2

Rohan has been served and removed from the queue.

Enter your choice: 3

Customers currently waiting:

1. Priya

Enter your choice: 4

Exiting Customer Service System.

Group D

1. Implement tree traversal techniques to display hierarchical employee structures in a company.

```
#include <iostream>
#include <string>
using namespace std;
// Structure for an Employee Node in the tree
struct Employee {
    string name;
    Employee* left;   // Left child (e.g., subordinate 1)
    Employee* right;  // Right child (e.g., subordinate 2)
};
// Function to create a new employee node
Employee* createEmployee(string name) {
    Employee* newEmp = new Employee;
    newEmp->name = name;
    newEmp->left = nullptr;
    newEmp->right = nullptr;
    return newEmp;
}
// Preorder Traversal (Root → Left → Right)
void preorder(Employee* root) {
    if (root == nullptr) return;
    cout << root->name << " ";
    preorder(root->left);
```

```
    preorder(root->right);
}
// Inorder Traversal (Left → Root → Right)
void inorder(Employee* root) {
    if (root == nullptr) return;
    inorder(root->left);
    cout << root->name << " ";
    inorder(root->right);
}
// Postorder Traversal (Left → Right → Root)
void postorder(Employee* root) {
    if (root == nullptr) return;
    postorder(root->left);
    postorder(root->right);
    cout << root->name << " ";
}
int main() {
    // Example company hierarchy
    /*
              CEO
             /   \
         Manager1  Manager2
          /  \       \
      EmpA   EmpB     EmpC
    */
```

```
    Employee* CEO = createEmployee("CEO");
    CEO->left = createEmployee("Manager1");
    CEO->right = createEmployee("Manager2");

    CEO->left->left = createEmployee("EmpA");
    CEO->left->right = createEmployee("EmpB");
    CEO->right->right = createEmployee("EmpC");

    cout << "\n--- Employee Hierarchy Tree Traversal ---\n";

    cout << "\nPreorder Traversal (Top-down order):\n";
    preorder(CEO);

    cout << "\n\nInorder Traversal (Left-root-right):\n";
    inorder(CEO);

    cout << "\n\nPostorder Traversal (Bottom-up order):\n";
    postorder(CEO);

    cout << "\n";
    return 0;
}
```

Output :

--- Employee Hierarchy Tree Traversal ---

Preorder Traversal (Top-down order):

CEO Manager1 EmpA EmpB Manager2 EmpC

Inorder Traversal (Left-root-right):

EmpA Manager1 EmpB CEO Manager2 EmpC

Postorder Traversal (Bottom-up order):

EmpA EmpB Manager1 EmpC Manager2 CEO

2. Implement BFS to find the shortest route between two locations in a road network.

```
#include <iostream>
#include <queue>
#include <map>
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>
using namespace std;
// Function to perform BFS and find the shortest path
void findShortestRoute(unordered_map<string, vector<string>> &graph, string start, string end) {
    queue<string> q;
    unordered_map<string, bool> visited;
    unordered_map<string, string> parent;  // To track the path

    q.push(start);
    visited[start] = true;
    parent[start] = "";
    bool found = false;
    while (!q.empty()) {
        string current = q.front();
        q.pop();

        if (current == end) {
```

```
            found = true;
            break;
        }
    for (auto neighbor : graph[current]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                parent[neighbor] = current;
                q.push(neighbor);
            }
        }
    }
    if (!found) {
        cout << " No route found between " << start << " and " << end << ".\n";
        return;
    }

    // Reconstruct shortest path
    vector<string> path;
    string temp = end;
    while (temp != "") {
        path.push_back(temp);
        temp = parent[temp];
    }
    reverse(path.begin(), path.end());
// Display shortest route
```

```
    cout << "\n Shortest route from " << start << " to " << end << ":\n";
    for (size_t i = 0; i < path.size(); i++) {
        cout << path[i];
        if (i != path.size() - 1)
            cout << " -> ";
    }
    cout << endl;
}
int main() {
    unordered_map<string, vector<string>> graph;
    int n;
 cout << "Enter number of roads (connections): ";
    cin >> n;
    cin.ignore();
 cout << "Enter the road connections (location1 location2):\n";
    for (int i = 0; i < n; i++) {
        string a, b;
        cin >> a >> b;
        graph[a].push_back(b);
        graph[b].push_back(a);  // Since roads are bidirectional
    }
  string start, end;
    cout << "\nEnter starting location: ";
    cin >> start;
    cout << "Enter destination location: ";
```

```
    cin >> end;
findShortestRoute(graph, start, end);
 return 0;
}
```

Output :

```
Enter number of roads (connections): 6
Enter the road connections (location1 location2):
A B
A C
B D
C D
D E
E F
Enter starting location: A
Enter destination location: F
```

Group E

1. A university stores student roll numbers in a sorted list. Implement binary search to find whether a given roll number exists in the list.

```
#include <iostream>
using namespace std;
// Function to perform binary search
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
      if (arr[mid] == key)
            return mid; // Roll number found
        else if (arr[mid] < key)
            low = mid + 1; // Search right half
        else
            high = mid - 1; // Search left half
    }
    return -1; // Roll number not found
}
int main() {
    int n, key;

    cout << "Enter number of students: ";
    cin >> n;
```

```
    int rollNumbers[100];
    cout << "Enter " << n << " roll numbers in sorted order:\n";
    for (int i = 0; i < n; i++) {
        cin >> rollNumbers[i];
    }
cout << "Enter roll number to search: ";
    cin >> key;
 int result = binarySearch(rollNumbers, n, key);
 if (result != -1)
        cout << " Roll number " << key << " found at position " << result + 1 <<
".\n";
    else
        cout << " Roll number " << key << " not found in the list.\n";
   return 0;
}
```

Output :

Enter number of students: 6

Enter 6 roll numbers in sorted order:

101 104 107 110 115 120

Enter roll number to search: 110

Roll number 110 found at position 4.

Enter roll number to search: 108

Roll number 108 not found in the list.

2. Sort a dataset of e-commerce product prices using quick sort.

```
#include <iostream>
using namespace std;
// Function to swap two elements
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
// Partition function for quick sort
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // choose the last element as pivot
    int i = low - 1;
   for (int j = low; j < high; j++) {
        if (arr[j] < pivot) { // smaller element
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}
// Quick Sort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
```

```
        int pi = partition(arr, low, high);
        // Recursively sort left and right parts
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
int main() {
    int n;
    cout << "Enter number of products: ";
    cin >> n;
  int prices[100];
    cout << "Enter product prices:\n";
    for (int i = 0; i < n; i++) {
        cin >> prices[i];
    }
 quickSort(prices, 0, n - 1);
 cout << "\n Sorted product prices (ascending order):\n";
    for (int i = 0; i < n; i++) {
        cout << prices[i] << " ";
    }
 cout << endl;
    return 0;
}
```

Output :

Enter number of products: 6

Enter product prices:

450 1200 999 250 750 300

Sorted product prices (ascending order):

250 300 450 750 999 1200

3 .Implement the Bubble Sort algorithm to sort an array of student marks.

```
#include <iostream>
using namespace std;
// Function to perform Bubble Sort
void bubbleSort(int marks[], int n) {
    for (int i = 0; i < n - 1; i++) {
        bool swapped = false; // To optimize if no swap happens in a pass
        for (int j = 0; j < n - i - 1; j++) {
            if (marks[j] > marks[j + 1]) { // Swap if elements are in wrong order
                int temp = marks[j];
                marks[j] = marks[j + 1];
                marks[j + 1] = temp;
                swapped = true;
            }
        }
        // If no swapping occurred in this pass, the array is already sorted
        if (!swapped)
            break;
    }
}
int main() {
    int n;
    cout << "Enter number of students: ";
```

```
    cin >> n;
 int marks[100];
    cout << "Enter marks of " << n << " students:\n";
    for (int i = 0; i < n; i++) {
        cin >> marks[i];
    }
bubbleSort(marks, n);
 cout << "\n Student marks in ascending order:\n";
    for (int i = 0; i < n; i++) {
        cout << marks[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Output :

Enter number of students: 5

Enter marks of 5 students:

78 45 89 60 55

Student marks in ascending order:

45 55 60 78 89

4. Implement the Insertion Sort algorithm to sort student names in ascending order

```
#include <iostream>
#include <string>
using namespace std;
// Function to perform Insertion Sort on strings
void insertionSort(string names[], int n) {
    for (int i = 1; i < n; i++) {
        string key = names[i];
        int j = i - 1;
 // Move names that are greater than key to one position ahead
        while (j >= 0 && names[j] > key) {
            names[j + 1] = names[j];
            j--;
        }
        names[j + 1] = key;
    }
}
int main() {
    int n;
    cout << "Enter number of students: ";
    cin >> n;
 string names[100];
    cout << "Enter " << n << " student names:\n";
```

```
    for (int i = 0; i < n; i++)
{cin >> names[i];
    }
insertionSort(names, n);
cout << "\n Student names in ascending (A–Z) order:\n";
    for (int i = 0; i < n; i++) {
        cout << names[i] << endl;
    }
    return 0;
}
```

Output :

Enter number of students: 5

Enter 5 student names:

Ravi

Anita

Kiran

Meena

Balu

Student names in ascending (A–Z) order:

Anita

Balu

Kiran

Meena

Ravi